ARMY RESEARCH LABORATORY

ARL

Computing Optic Flow with ArduEye Vision Sensor

by Kathryn Schneider, Dr. Joseph Conroy, and Dr. William Nothwang

ARL-TR-6292 January 2013

Approved for public release; distribution unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-6292 January 2013

Computing Optic Flow with ArduEye Vision Sensor

Kathryn Schneider, Dr. Joseph Conroy, and Dr. William Nothwang
Sensors and Electron Devices Directorate, ARL

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* January 2013	**2. REPORT TYPE** Final		**3. DATES COVERED (From - To)**
4. TITLE AND SUBTITLE Computing Optic Flow with ArduEye Vision Sensor			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kathryn Schneider, Dr. Joseph Conroy, and Dr. William Nothwang			**5d. PROJECT NUMBER**
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-SER-L 2800 Powder Mill Road Adelphi, MD 20783-1197			**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-6292
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			
13. SUPPLEMENTARY NOTES			

14. ABSTRACT

Flight control of micro aerial vehicles (MAVs) is challenging due to the size of small-scale robotics platforms, which places size, weight, and power limitations on onboard sensors, as well as restricts the data processing algorithms that can be implemented on the platforms. We focus on computing optic flow in order to control a MAV using small vision chips as opposed to a complex camera system in order to cut down on the size and weight of the sensors as well as the processing power needed. We propose using an externally produced vision sensor by Centeye, ArduEye, to develop an algorithm for computing optic flow, from existing algorithms tailored to the sensor, to be used to control a quadrotor. Such efforts will benefit the various MAV research thrusts by providing a successful optic flow sensor and data processing algorithm that can be applied to the flight control of other robotic platforms.

15. SUBJECT TERMS

Optical flow, ArduEye, vision based navigation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** William D. Nothwang
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			
Unclassified	Unclassified	Unclassified	UU	26	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-1163

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction/Background

Micro aerial vehicles (MAVs) have been the focus of many research thrusts in recent years with regard to autonomous robotics used in surveillance, reconnaissance, and other observational tasks. Being able to perform these tasks with unmanned robotic vehicles would allow situational and environmental awareness without putting any humans in harm's way. Furthermore, they extend the usefulness of unmanned robotic vehicles. MAVs can maneuver in complex spaces and are less detectable than larger platforms (*1*).

The first step in developing a MAV that can do any of these tasks is flight control. Biological research has shown many sensory components of flight control in insects, one of which is using visual data to maintain a straight path without running into objects (*2*). Studies of insects' flight response with respect to their visual field have found that when an insect senses a moving stimulus, they turn towards that stimulus in order to reduce the motion of their visual field and stabilize their orientation with respect to the environment (*2*). This response can be recreated in a robotic platform through the computation and reduction of optic flow.

Due to the size of small-scale robotics, there are size, weight, and power limitations on the onboard sensors that can be implemented on the robotic platforms. There is a significant need for small, light, less power-hungry sensors and sensory data processing algorithms in order to control the robotic platform within the constraints of the platform itself. Due to these constraints, using a complex camera system would be impossible, so some other vision sensor and accompanying algorithm must be implemented to compute optic flow. We propose using an externally developed and produced vision sensor by Centeye to develop an algorithm for computing optic flow, from existing algorithms, to be used to control a quadrotor. Such efforts will benefit the various MAV research thrusts by providing a successful optic flow sensor and data processing algorithm that can be shared and applied to other robotic platforms to aid with other research.

While the overarching goal of this project is to create a working algorithm for the computation of optic flow for multiple vision sensors to control the flight of a quadrotor, this project's focus is on characterizing the vision sensors themselves and developing an algorithm that is optimized for the sensor. In initial testing of the sensor, we found a clear tradeoff between resolution of the image acquired from the sensor and rate of acquisition, thus changing the focus of the algorithm created for the sensor. The algorithm now computes optic flow from Centeye's vision sensor for varying image resolutions to ultimately create a dynamic algorithm that creates an efficient tradeoff between frame rate and resolution. In doing this, we prove that trading off between frame rate and resolution is viable given a dynamic speed of the vehicle on which the sensors are mounted and find the appropriate ranges of vehicle speeds for such tradeoffs. Our hypothesis is that the high resolutions will reliably detect low speeds but, due to the slow frame rate, will not be able to refresh fast enough to accurately detect higher speeds. Low resolutions, on the other hand, have a frame rate fast enough to detect low and high speeds, but the low resolutions will likely cause a lot of noise in the optic flow output, making them inaccurate at lower speeds.

1.1 Optic Flow

Optic flow is an approximation of the apparent image motion caused by the motion of the observer relative to the visual scene. Its computation is based upon local derivatives in a sequence of images. In two dimensional (2-D), optic flow specifies how much each pixel of an image moves between adjacent images, and in three dimensional (3-D), it specifies how much each volume voxel (volumetric pixel) moves between adjacent volumes (*3*).

The basis of differential optic flow is the motion constraint equation whose derivation is shown in Barron and Thacker's "Tutorial: Computing 2D and 3D Optical Flow" (*3*) and in less detail below. *I(x, y, t)* is a pixel at time *t* and position *(x, y)*. It moves by *δx* and *δy* in time *δt* to *I(x+ δx, y+ δy, t+ δt)*, as shown above in figure 1. *I(x, y, t)* and *I(x+ δx, y+ δy, t+ δt)* are the same pixel and are therefore equal, giving rise to the following equation:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{1}$$

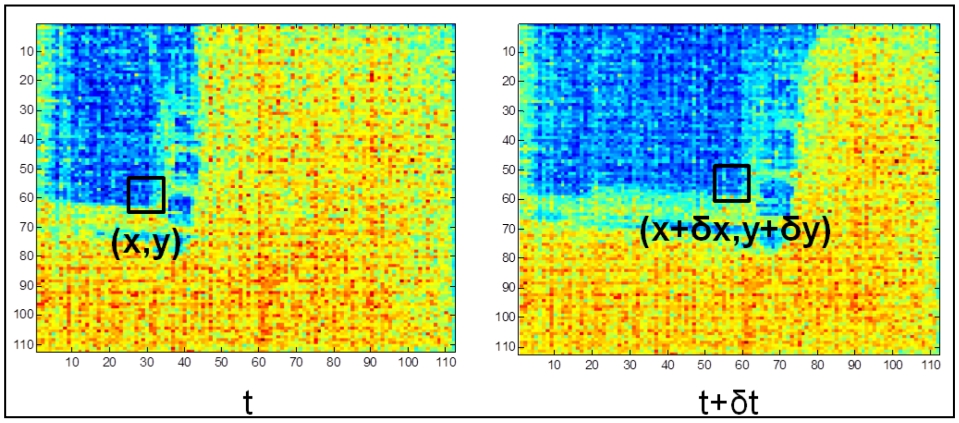


Figure 1. Optic flow illustration.

The assumption that the images are the same is true to a first approximation if *δx, δy,* and *δt* are not too large. A first-order Taylor series expansion about *I(x, y, t)* is performed to define *I(x+ δx, y+ δy, t+ δt)*.

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \cdots \tag{2}$$

The higher-order Taylor terms are small enough to discard. Since *I(x, y, t)* and *I(x+ δx, y+ δy, t+ δt)* are equal they cancel each other out in equation 2. We can then take the remaining equation and divide by *δt*.

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

$$\frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} = 0 \qquad (3)$$

The partial derivatives *∂I/∂x*, *∂I/∂y*, and *∂I/∂t* can be written as I_x, I_y, and I_t, the intensity derivatives. *δx/δt* and *δy/δt* can be written as v_x and v_y, the *x* and *y* components of optic flow. These substitutions give rise to the following equations.

$$I_x v_x + I_y v_y + I_t = 0$$

$$\left(I_x, I_y\right) \cdot \left(v_x, v_y\right) = -I_t$$

$$\nabla I \cdot \vec{v} = -I_t \qquad (4)$$

The final equation is the 2-D Motion Constraint Equation, where $\nabla I = \left(I_x, I_y\right)$is the spatial intensity gradient and $\vec{v} = (v_x, v_y)$ is the image velocity or optic flow at pixel (*x*, *y*) at time *t*.

Optic flow algorithms calculate the image velocity or optic flow vector $\vec{v}$ shown in the 2-D Motion Constraint Equation (equation 4). Optic flow can be computed by feature tracking (tracking the motion of corresponding features in adjacent images), comparing local gradients of image intensity in space and time, or comparing the temporal frequencies in various regions of the spatiotemporal frequency spectrum (*5*). Optic flow algorithms have been developed for each of these approaches and the two that are tested using Centeye's vision sensor are discussed.

1.2 Lucas Kanade Algorithm

The Lucas–Kanade (LK) method is a differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade, which assumes a constant model for $\vec{v}$ in a small spatial neighborhood and solves the Motion Constraint Equation for all the pixels in that neighborhood by implementing a local least-squares approximation (*4*). LK takes the 2-D Motion Constraint Equation and converts it into matrix form. That matrix form is shown as follows:

$$(\nabla I)v = -I_t \qquad (5)$$

This equation is then solved using least-squares approximation.

$$(\nabla I)^T(\nabla I)v = (\nabla I)^T(-I_t)$$

$$v = \{(\nabla I)^T(\nabla I)\}^{-1}(\nabla I)^T(-I_t) \tag{6}$$

This method of computing optic flow has also been expanded upon and improved through the use of pyramid interpolation and hierarchical methods for more precise object tracking. The LK algorithm is one of the most commonly used algorithms for computing optic flow in computer vision due to its assumption that the optical flow in a small neighborhood of pixels is the same, allowing for a simplification of a least-squares approximation (*9*).

1.3 Image Interpolation Algorithm

Srinivasan proposes an algorithm (the image interpolation algorithm [IIA]) for computing optic flow, which does not require the tracking of features or measurement of image velocities at many different locations. Rather IIA estimates optic flow by a non-iterative process, which interpolates the position of the moving image with respect to a set of reference images (*6*). IIA estimates the motion of a plane that is capable of translation along the *x*- and *y*-axes, which are perpendicular to the camera axis, and rotate about an arbitrary axis perpendicular to the plane. The motion estimate of the plane is done within a patch of the plane, which is defined by a window function. The following equations used for the computation of optic flow in IIA come from Srinivasan's "An Image-Interpolation Technique for the Computation of Optic Flow and Egomotion" (*6*).

Intensity functions of two images in time from t_0 to t are denoted by $f_0\ (x,y,t_0)$ and $f(x,y,t)$. IIA interpolates f from f_0 and the references images f_1, f_2... f_6. These reference images are defined as versions of f_0 by shifts in the x, y, and θ directions.

$$f_1(x,y) = f_0(x + \Delta x_{ref})$$

$$f_2(x,y) = f_0(x - \Delta x_{ref})$$

$$f_3(x,y) = f_0(y + \Delta y_{ref})$$

$$f_4(x,y) = f_0(y - \Delta y_{ref})$$

$$f_5(x,y) = f_0(x', y')$$

$$f_6(x,y) = f_0(x'', y'') \tag{7}$$

where the x', x'', y', and y'' expressions are as follows:

$$x' = x \cdot sin\Delta\theta_{ref} + y \cdot cos\Delta\theta_{ref}$$

$$x'' = x \cdot sin(-\Delta\theta_{ref}) + y \cdot cos(-\Delta\theta_{ref})$$

$$y' = x \cdot cos\Delta\theta_{ref} + y \cdot sin\Delta\theta_{ref}$$

$$y'' = x \cdot cos(-\Delta\theta_{ref}) + y \cdot sin(-\Delta\theta_{ref}) \tag{8}$$

We assume that *f* can be approximated by$\hat{f}$:

$$\hat{f} = f_0 + 0.5\left(\frac{\widehat{\Delta x}}{\Delta x_{ref}}\right)(f_2 - f_1) + 0.5\left(\frac{\widehat{\Delta y}}{\Delta y_{ref}}\right)(f_4 - f_3) + 0.5\left(\frac{\widehat{\Delta\theta}}{\Delta\theta_{ref}}\right)(f_6 - f_5) \tag{9}$$

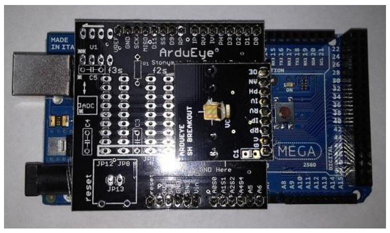
This assumption is true if the displacement of the reference images f_1, f_2... f_6 and image *f* relative to the original image f_0 are small compared to the highest spatial-frequency component's spatial wavelength in the image. IIA is not as computationally complex as LK, but rather takes its inspiration from biology to linear combine shifted versions of the original image captured to compute optic flow.

1.4 Centeye's ArduEye Vision Sensor

Centeye is a corporation started by Geoffrey Barrows, which specializes in embedded vision systems for robotics applications (*7*). Centeye develops software and hardware for ArduEye, which is an open source project for the implementation of vision sensors using an Arduino microcontroller (*8*). These vision sensors or "vision chips" are similar to regular charge-coupled device (CCD) or complementary metal-oxide semiconductor (CMOS) imaging chips, which convert an optical image into electrical signals but unlike the CCD and CMOS chips, which digitize the image, a vision chip performs image processing on the raw image using a mixture of analog and digital circuitry (*7*).

The output of the vision chip is a pre-processed version of the image; therefore, the resolution of the image can be reduced during acquisition as opposed to performing post-acquisition reduction on the Arduino, saving processing time and memory. This pre-processing property of the ArduEye sensor was exploited so that we could specify the number of pixels binned in order to decrease the resolution of the image acquired. In this way, only every 2, 4, 8, or 16 pixels were acquired to represent the "super pixel" of the 2x2, 4x4, 8x8, or 16x16 sections. The resolutions of the acquired image after this pre-processing are 56x56, 28x28, 14x14, and 7x7, respectively.

The ArduEye chip used in this project is from the Stonyman series, which was connected to the Arduino Mega 2560 board using the ArduEye Rocket System breakout board, all of these components are shown in figure 2. Two Stonyman chips were initially tested to determine which one would be used for the project. One chip uses pinhole optics, which finds the brightest pixel in the image and acquires a 16x16 image centered at that pixel, and the other chip uses cell phone optics, which can acquire up to a 112x112 image. Both of these chips are shown in figure 2; the cell phone optics chip is on top and pinhole optics chip is on the bottom. Because the eventual goal of this project is flight control of a quadrotor, the cell phone optics chip was used to acquire a larger image for more precise optic flow measurements.



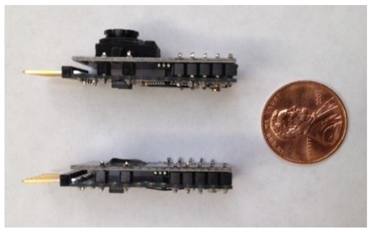


Figure 2. ArduEye vision chip on Stonyman breakout board connected to Arduino Mega (*8*) (left) and the Stonyman vision chips (*7*) (right).

Much of the code used in this project, such as image acquisition and optic flow computation, was written by Centeye and put out as open source downloadable code from their Web site (*8*). Since all of this code is available, it is not specifically discussed in this report, but rather is mentioned with the description of the optic flow algorithm in section 2.

2. Experiment/Calculations

The optic flow algorithm implemented is a simple loop that acquires the current image from the vision sensor and then uses that image and the previously acquired image to compute optic flow. That high level loop is shown in figure 3. The variability in the algorithm comes with the bin number, which determines the resolution of the image acquired from the chip and the switch between using LK or IIA to compute optic flow. This variability in the algorithm is illustrated in figure 3 as variables that are passed in to the functions within the high level task pictured. All the functions used to acquire the image from the sensor and compute optic flow are taken from the ArduEye Web site (*8*). The only additions to the code are the implementation of those functions in a simple loop and the printing of the data to conform to the LabView data acquisition system used for the experiments conducted on the rate table. The LK and IIA functions were written by Centeye and implement the equations presented in section 1 using pointers; those functions can be found on the ArduEye Web site (*8*). Additional functions were written for initial testing but were not used in the final algorithm. Those functions are discussed with the first experiment.

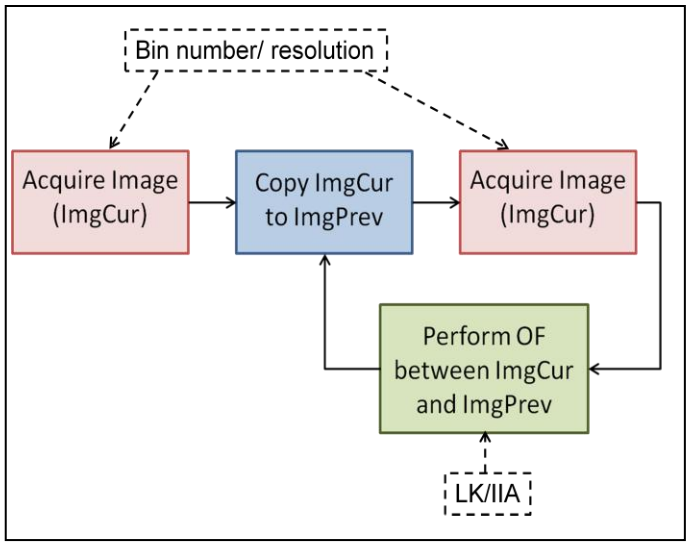


Figure 3. Optic flow algorithmic loop.

Two experiments were conducted within this project to provide initial characterization data of the sensors and their performance. The first experiment attains the Arduino's processing times needed to acquire the image from the sensor and compute optic flow for each image resolution previously discussed. This initial timing experiment is conducted to determine the performance of the Arduino, which dictates the final algorithm. The second experiment determines the accuracy of optic flow computations for each image resolution at varying speeds by performing rate table experiments. The specifics of these experiments are discussed in sections 2.1 and 2.2.

2.1 Timing Experiment

This experiment uses only the Arduino software and the ArduEye vision sensor to determine the processing time of functions used in the optic flow algorithm. These times are acquired using the Arduino function millis(), which returns number of milliseconds since the Arduino began running the current program. This time is reported after each run of the main loop in the program, and thus the timing data are taken as an average of the difference between the times reported by millis() on the serial monitor.

2.2 Rate Table Experiment

The rate table experiment uses the optic flow algorithm to estimate angular velocity. The ArduEye sensor is mounted on a rate table and spun at various rates to characterize the noise and sensitivity the sensor at each resolution using both LK and IIA. The algorithm's estimate of angular velocity is compared with the actual rate at which the table is being driven to determine noise and sensitivity. Since translation velocity is held to zero with the sensor mounted at the center of the rate table, optic flow is taken as a direct correlation to angular velocity. This can be realized through equation 10 (*1*) in which *OF* is optic flow, ω is angular velocity, *v* is translational velocity, and *D* is the distance of the sensor to an object.

$$OF = -\omega + \frac{v}{D} \tag{10}$$

When translational velocity *v* is zero, optic flow is proportional to angular velocity ω. In this way, the rate table experiment is a simple way to reduce the number of variables in the optic flow equation 10 and characterize the noise and sensitivity of the sensor.

Figure 4 shows a picture of the experimental setup. For a rate table, we used a motor mounted under a test stand, which rotates the metal bar attached to the stand. We then attached the Arduino board with the ArduEye sensor to the bar and read data off of the Arduino to a computer running LabView. The LabView program then compiled a text file of the optic flow values sent by the Arduino and the time stamp at which those values were reported. The speed of the motor was controlled through a laptop, which told the motor when to start and stop and at what speed to rotate.

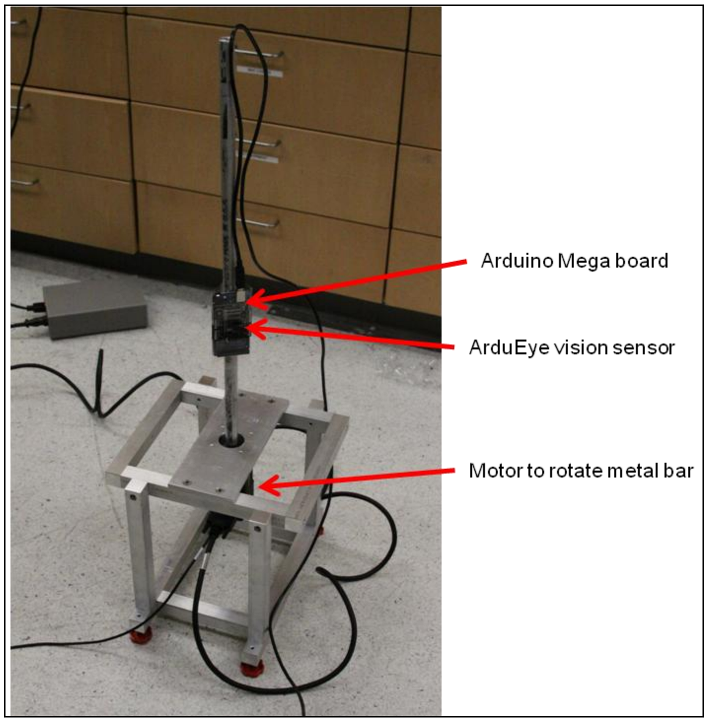


Figure 4. Rate table experimental setup.

3. Results and Discussion

3.1 Timing Experiment

In initial testing of the optic flow algorithm, we found that the Arduino Mega board could not acquire and save the 112x112 image in its flash memory and therefore another function was written to acquire and save the image from the vision sensor to the EEPROM on the board. Another hardware constraint found was that after acquiring an image and saving it either to flash memory or EEPROM, resolutions higher than 28x28 could not be passed into the optic flow functions (both LK and IIA) without overflowing the Arduino flash memory and causing the entire program to crash. To deal with this constraint, we downsized the acquired image by averaging across 2x2 and 4x4 sections to acquire averaged “super pixels,” as opposed to the binned “super pixels” acquired originally, which sample one pixel to represent a group of pixels. Both the EEPROM and averaged “super pixel” added functions became part of the timing experiment.

Figure 5 shows the results of the timing experiment for all five discussed resolutions with the following four functions: acquiring and saving the image from the vision sensor to the EEPROM, saving the image to the flash memory, performed the LK optic flow computation, and performing the IIA computation. The LK and IIA processing times overlay one another on the graph.

The first immediate result we can see from this graph is that by increasing the resolution by a window size factor of 2 each function’s time increases by a factor of around 4, which is to be expected given that an increase by 2 in the window size is a increase by 4 in the amount of pixels in the image. We can also see that although saving to the EEPROM takes significantly more time than saving to the flash memory (an increase by a factor of between 8 and 23), saving to the EEPROM is the only option for acquiring the full 112x112 resolution. Similarly, IIA and LK can only be computed for 7x7, 14x14, and 28x28 resolutions.

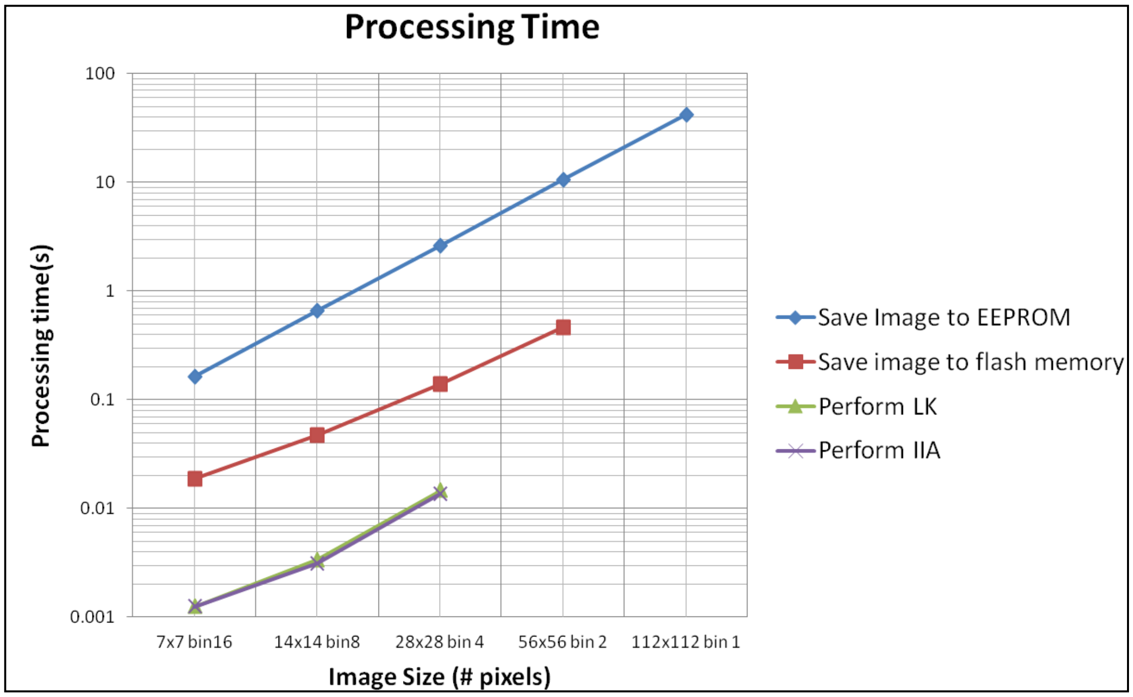


Figure 5. Arduino's processing times.

The results of the timing experiment are shown in figure 6 for the averaging "super pixel" solution to the size resolution constraints of the optic flow functions. We can see from this graph that even with 4x4 pixel averaging, the greatest resolution attainable is 56x56, so the entire 112x112 resolution cannot be downsized in order to have optic flow computed. This information was useful to identify the limitations of the Arduino microcontroller.

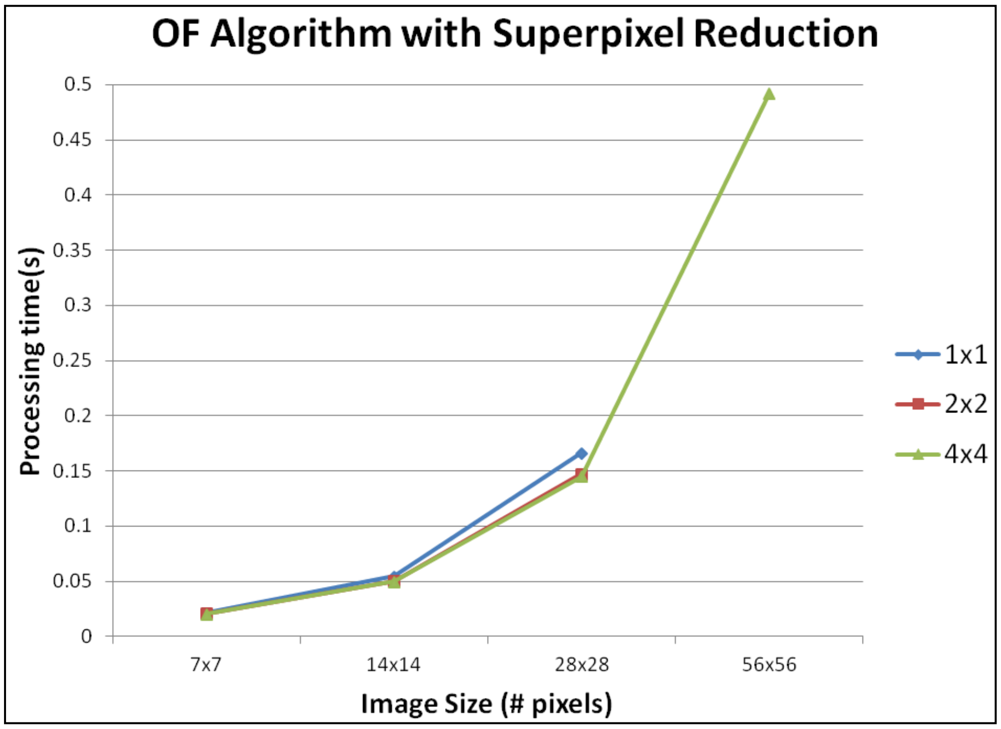


Figure 6. "Super pixel" reduction processing times.

The conclusion drawn from these timing experiments is that the Arduino's memory cannot handle higher resolutions without reduction using "super pixels," and even with that fix, the highest resolutions cannot be processed. In order to move forward with this project and use the ArduEye vision sensors for flight control of a quadrotor we need to move toward a microcontroller with more memory and processing power.

3.2 Rate Table Experiment

The results of the timing experiments restricted the image resolutions that could be tested on the rate table to 7x7, 14x14, and 28x28 pixels. Each resolution was run at six different speeds, both positive and negative, and at rest. Each speed was maintained for a variable amount of time, speeds were switched either once we decided enough data were collected or if the speed could not be maintained anymore due to physical constraints such a the input/output (I/O) wire over twisting. These tests were done for both the LK and IIA algorithm to determine the differences in the sensor's noise and sensitivity between each of those algorithms.

Figures 7 and 8 show the raw results for both LK and IIA at all resolutions and speeds. These speeds are slightly different between the LK and IIA tests because the motor speed controller proved to be slightly variable and exact same speeds could not be accomplished. These graphs show the optic flow values as computed by the Arduino. The *x*-axis represents time but as the indices of the optic flow data in an array, not as the actual time stamp of the time at which the optic flow value was recorded. The rate at which the table was rotating in radians per second is noted above or below the section of the graph to which it corresponds. The six speeds were broken into two continuous tests in which three speeds (both positive and negative) were tested in succession.

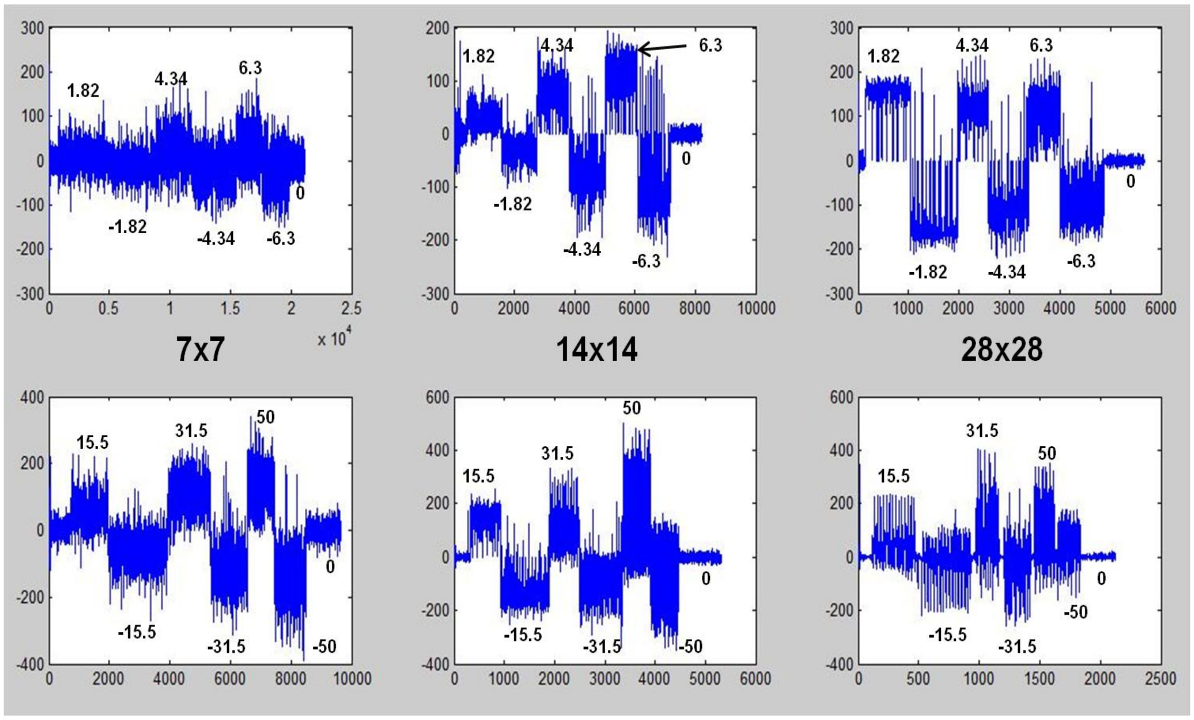


Figure 7. IIA rate table raw data.

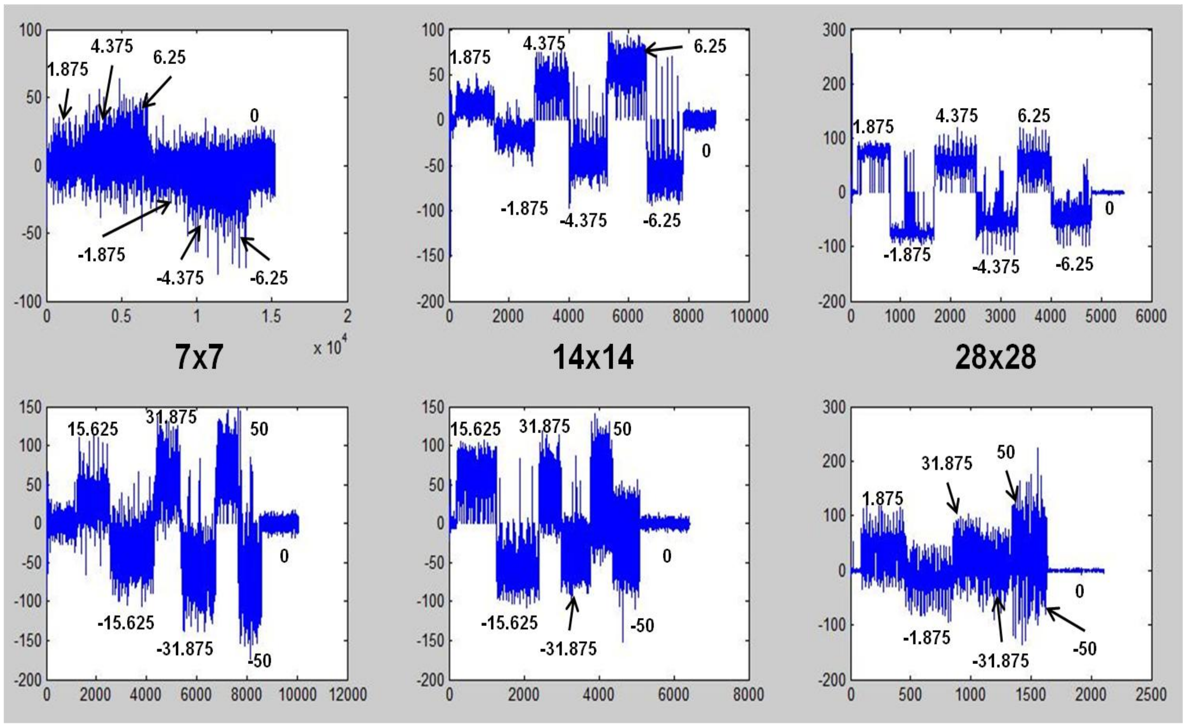


Figure 8. LK rate table raw data.

The first conclusion drawn from this data was that the rate of image acquisition for each resolution agreed with the times attained in the timing experiment. These times were attained from the raw data by averaging the difference in the times that the optic flow was obtained. Thus, the frame rates of the 7x7, 14x14, and 28x28 resolutions were confirmed to be ~45, ~18, and ~6 Hz, respectively.

To analyze these raw data, the mean of the optic flow for each rate was computed as well as the standard deviation. The means and standard deviations were taken across a window of the speed's section of data such that the same number of data points were being used to calculate the mean and standard deviation for each speed and resolution. The number of data points in each window was determined by the smallest window that number of points was 150. These data were then scaled to reflect the actual rate at which the table was being driven to yield the graphs shown in figures 9 and 10. The means are graphed as a function of the rate at which the table was being driven and the standard deviations are shown as the error bars associated with each optic flow mean.

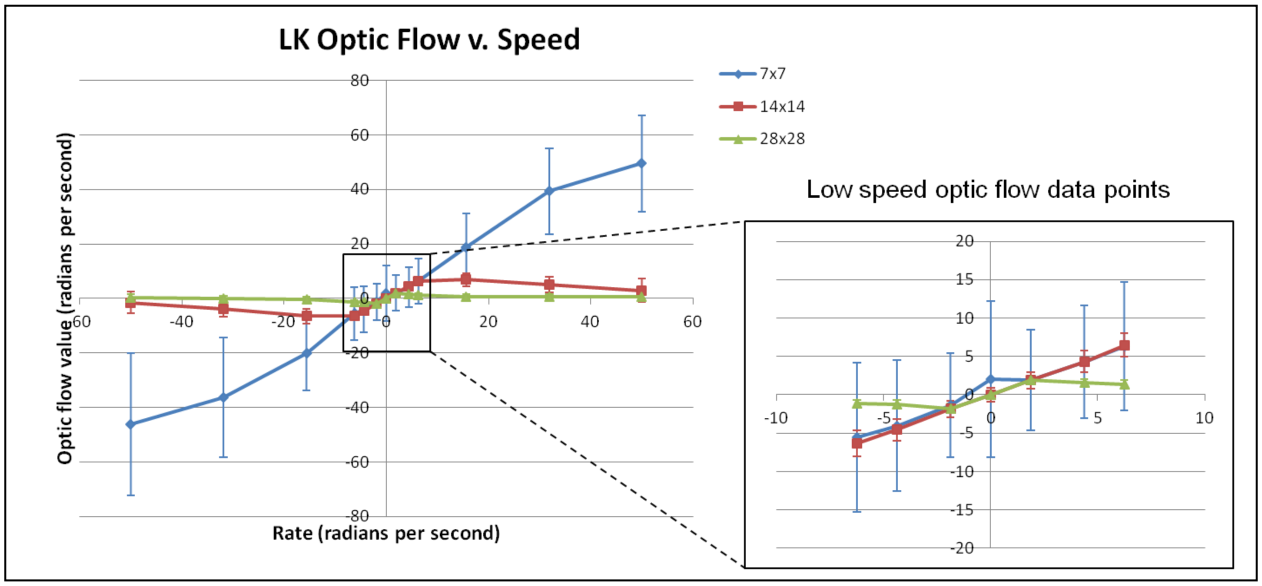


Figure 9. Lucas-Kanade (LK) Optic flow vs. speed, with low speeds data points shown in blow up graph.

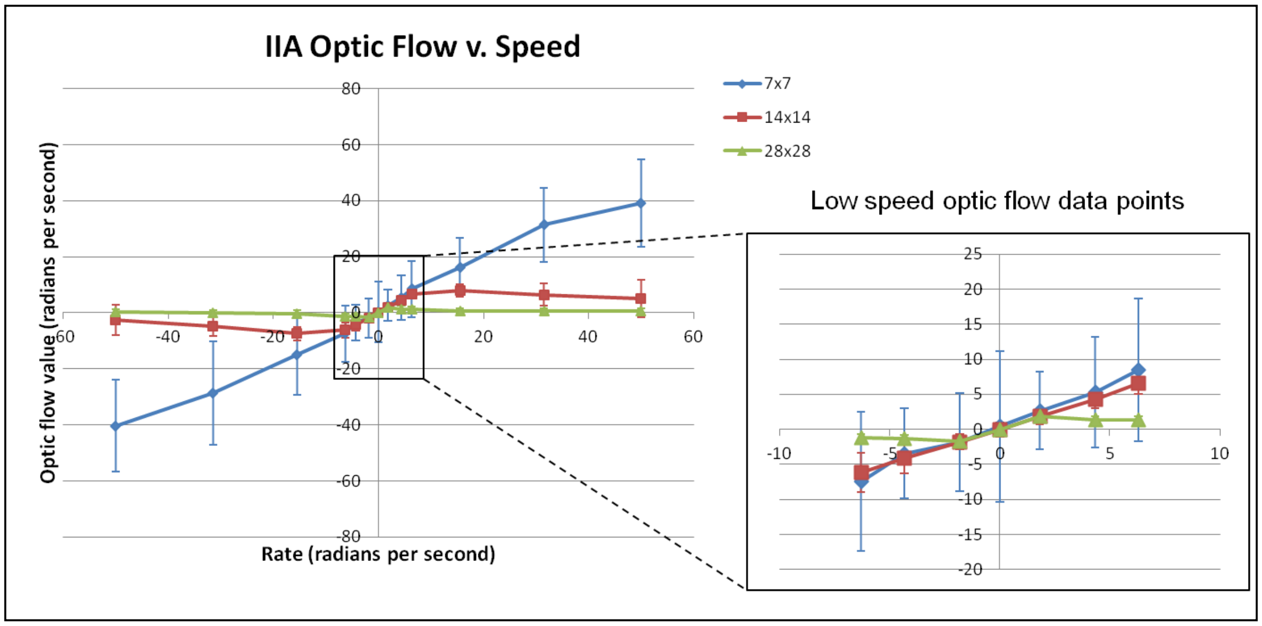


Figure 10. Image Interpolation Algorithm (IIA) optic flow vs. speed, with low speeds data points shown in blow up graph.

The 7x7 optic flow for both LK and IIA exhibits a fairly linear trend with the rate at which the sensor is being driven, demonstrating that the low resolution has a large range of high sensitivity. However, a lot of noise is seen within that entire range. The higher resolutions of 14x14 and 28x28 have significantly less noise but are only high sensitivity at low speeds. The graphs of the blown up lower speeds show that all resolutions tested are accurate for speeds less in magnitude than ~2 radians per second, but outside that range 28x28 drops off significantly in sensitivity.

The 14x14 resolution sees a similar drop off in sensitivity in speeds greater in magnitude than ~10 radians per second. These results confirm our hypothesis that as the rate at which the sensor is being driven increases, the higher resolutions' slower frame rate cannot capture images often enough to get an accurate estimate of optic flow. Although the 7x7 frame rate can acquire images fast enough, the lower resolution introduces a lot of noise into the system due to the smaller number of pixels used to represent the entire image. The higher resolutions do not introduce as much noise due to the greater number of pixels giving a better view of the actual image that the vision sensor sees.

The outlier in these results is the data from rest. For all resolutions, the zero speed has higher noise then the neighboring low speeds and for 7x7 LK that zero value appears as a value of ~2 radians per second whereas the other resolutions yield a value much closer to 0 radians per second. This bias in the LK 7x7 data may be due to the way in which data were collected and where the vision sensor was pointing when the zero data were collected. Optic flow is a measure of image velocity so if while the sensor was at rest its image field changed, that would read as non-zero optic flow. The reasoning could also explain the high noise associated with the zero data. The change in the image field due to human movement or light changes can likely be neglected when the sensor is being driven at a non-zero rate. This is because the rate at which the sensor is moving has more of an impact on the optic flow output than the ambient movement of the environment. This, however, is not true at rest and may account for the bias seen in that data.

The conclusion we can draw from these data is that there is a clear tradeoff between frame rate and resolution, which can be exploited to optimize sensitivity and noise relative to the speed at which the sensor is moving.

4. Conclusions

The results of the timing experiments revealed that the Arduino's memory cannot handle higher resolutions higher than 28x28 pixels; therefore, we need to move towards a microcontroller with more memory and processing power. The next step in this project will be to test the ArduEye sensors on a processor such as the ARM. The same characterization experiments will take place with that board.

The rate table results proved that the low resolution image has a large range of high sensitivity for optic flow, but a lot of noise is seen within that entire range. Higher resolutions have significantly less noise but are only highly sensitivity at low speeds. These results were as expected and the same results should come when the ArduEye sensor is tested with the ARM processor. If frame rate can be sped up for the higher resolutions using this processor, these results can be improved such that the higher resolutions have a higher sensitivity over a greater range of speeds. Also, with a processor with greater memory, the even higher resolutions of 56x56 and 112x112 pixels can be tested.

This research proved that there is a clear tradeoff between frame rate and resolution, leading to either more sensitive results with high noise or less sensitivity with lower noise. We proved that at low speeds the higher resolutions of 28x28 and 14x14 pixels perform as well as the 7x7 low resolution but with less noise, whereas the sensitivities of those high resolutions drop off immediately after low speeds. Those drop-off speeds were determined to be ~2 radians per second in magnitude for 28x28 and ~10 radians per second in magnitude for 14x14. In this way, the high resolutions of 14x14 and 28x28 are just as accurate and more reliable for the lower speeds (~10 radians per second), but the 7x7 low resolution must be used for any speed outside of that range. Using these results as a proof of concept, a dynamic algorithm can be created that sets the resolution of the image being acquired according to the speed of the vehicle. More tests need to be performed before such an algorithm is created to test the sensitivity and noise of the sensor at varying translational velocities in additional to purely angular velocities.

The overall goal of this project is to integrate multiple ArduEye sensors on one platform for fully autonomous flight control, so future work needs to focus on how to combine the optic flow outputs of multiple sensors to translate those combined outputs into actuation. Flight control using optic flow must also take in account the dynamics of the vehicle being flown as well as the environment in which the flight is being attempted. All subsequent work will build off of the vision sensor characterization results from this project.

5. References

1. Green, W. E.; Oh, P. Y. Optic-Flow-Based Collision Avoidance. *IEEE Robotics & Automation Magazine* **2008**, 96–103.

2. Srinivasan, M. V.; Poteser, M. Motion Detection in Insect Orientation and Navigation. *Vis. Res.* **1999**, *29*, 2749–103.

3. Barron, J. L.; Thacker, N. A. *Tutorial: Computing 2D and 3D Optical Flow*; Tina Memo No. 2004-012; Manchester, M13 9PT, 2005.

4. Lucas, B. D.; Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. *DARPA Image Understanding Workshop* **1981**, 121–130.

5. University of Central Florida Department of EECS Computer Science Division. Lecture 17 Computing Optical Flow: Lucas & Kanade. http://www.cs.ucf.edu/courses/cap6411/cap5415/Lecture-16.PDF (accessed July 2012).

6. Srinivasan, M. V. An Image-Interpolation Technique for the Computation of Optic Flow and Egomotion *Biol. Cybern.* **1994**, *71*, 401–415.

7. Centeye, Inc. http://centeye.com (accessed June 2012).

8. ArduEye: An open source programmable vision sensor for Arduino. http://www.ardueye.com (accessed June 2012).

9. Duhamel, P. E.; Poter, J.; Finio, B.; Barrow, G.; Brooks, D.; Wei, G. Y.; Wood, R. Hardware in the Loop for Optical Flow Sensing in a Robotic Bee. *IEEE/RSJ International Conference on Intelligent Robots and Systems* **2011**, 1099–1106.

List of Symbols, Abbreviations, and Acronyms

2-D	two dimensional
3-D	three dimensional
CCD	charge-coupled device
CMOS	complementary metal-oxide semiconductor
I/O	input/output
IIA	image interpolation algorithm
LK	Lucas–Kanade
MAVs	micro aerial vehicles

1 (PDF only) DEFENSE TECHNICAL
INFORMATION CTR
DTIC OCA
8725 JOHN J KINGMAN RD
STE 0944
FORT BELVOIR VA 22060-6218

1 DIRECTOR
US ARMY RESEARCH LAB
IMAL HRA
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 DIRECTOR
US ARMY RESEARCH LAB
RDRL CIO LL
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 US ARMY RSRCH LAB
ATTN RDRL VTU V C KRONINGER
ABERDEEN PROVING GROUND MD
21005

1 US ARMY RSRCH LAB
ATTN RDRL-HRS-C K MCDOWELL
ABERDEEN PROVING GROUND MD
21005

15 US ARMY RSRCH LAB
ATTN RDRL CII A E STUMP
ATTN RDRL CII A J FINK
ATTN RDRL CII A S H YOUNG
ATTN RDRL CII A J TWIGG
ATTN RDRL CIN B SADLER
ATTN RDRL SED E B MORGAN
ATTN RDRL SER L A WICKENDEN
ATTN RDRL SER L B PIEKARSKI
ATTN RDRL SER L J PULSKAMP
ATTN RDRL SER L R POLCAWICH
ATTN RDRL SER L W NOTHWANG
ATTN RDRL SER L G SMITH
ATTN RDRL SER L J CONROY
ATTN RDRL SER P AMIRTHARAJ
ATTN RDRL SE SEDD DIRECTOR
ADELPHI MD 20783-1197

3 CALIFORNIA INSTITUTE OF
TECHNOLOGY
ATTN MC 107-81 R MURRAY
ATTN MC 136-93 S HAN
ATTN MC 305-16 E WOLFF
1200 E CALIFORNIA BLVD
PASADENA CA 91125

2 UNIVERSITY OF MARYLAND
AUTONOMOUS VEHICLE
LABORATORY
ATTN S HUMBERT,
3182 MARTIN HALL,
ATTN DANIEL SILVERSMITH, 6801
PREINKERT DR., UNIT 7312B
COLLEGE PARK MD 20742

2 UNIVERSITY OF MICHIGAN
DEPARTMENT OF ELECTRICAL
ENGINEERING AND COMPUTER
SCIENCE
ATTN E. YOON
ATTN SJ PARK
2400 EECS BLDG.,
1301 BEAL AVENUE
ANN ARBOR, MI 48109-2122

1 UNIVERSITY OF PENNSYLVANIA
MULTIMEDIA AND NETWORKING
LABORATORY
ATTN A KOPPEL
306 MOORE
200 SOUTH 33RD STREET
PHILADELPHIA, PA 19104

1 UNIVERSITY OF CALIFORNIA
BERKELEY
ATTN DAN CALDERONE
337 CORY HALL
BERKELEY, CA 94720-1772

1 HARVARD UNIVERSITY
ATTN NICHOLAS PERKONS,
509 KIRKLAND M.C.
95 DUNSTER ST.
CAMBRIDGE, MA 21038

1 CEDARVILLE UNIVERSITY
ATTN MICHAEL COMPARETTO
251 N. MAIN STREET # 2554,
CEDARVILLE, OH 45314

1 PRINCETON UNIVERSITY
YUAN CHEN
1471 FRIST CAMPUS CENTER
PRINCETON, NJ 08544

1 VISHNU GANESAN
2280 SOUTH OVERLOOK ROAD,
CLEVELAND OH, 44106

1 KESSHI JORDAN
14526 MACCLINTOCK DR.
GLENWOOD, MD 21738

1 KATHRYN SCHNEIDER
8204 BALTIMORE AVE
APT#1026C
COLLEGE PARK, MD 20740

1 MICHAEL ROBERTS
12608 IVYSTONE LANE
LAUREL, MD 20708

1 CORDELL REID
8531A GREENBELT RD,
APT 202,
GREENBELT MD 20770

1 TRISTAN HELMS
928 S STATE ST APT 4
ANN ARBOR, MI 48104

1 PDF NAVAL SURFACE WARFARE CTR
ATTN JESSE CAMPBELL, K74
5370 MARBLE RD STE 143
DAHLGREN, VA 22448-5165

TOTAL: 39 (2 PDF, 37 HCS)